Teaching Computer Science with Abstract Strategy Games

Dan Garcia (Moderator)
EECS Department
Univ. of California, Berkeley
Berkeley, CA, USA
ddgarcia@cs.berkeley.edu

Ivona Bezakova
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
ib@cs.rit.edu

Adam Blank
CMS Department
California Institute of Technology
Pasadena, CA, USA
blank@caltech.edu

Neal Terrell
Computer Science Department
Cal State University, Long Beach
Long Beach, CA, USA
neal.terrell@csulb.edu

ABSTRACT

Abstract Strategy Games are games of no chance with complete information – all players (usually two) know all there is about the current position; nothing is hidden. Examples of popular games are Tic-Tac-Toe, Chess, Checkers, Connect-4, Reversi, Mancala, Nim, Dots-and-Boxes, and Go; there are thousands more [5]. In addition to the cultural history and remarkably beautiful mathematics locked within the strategies and game trees, we have found they form a wonderfully fertile, rich, and engaging source of activities around which to teach fundamentals of computer science. This panel will explore the ways in which we have used these games with our students, through interactive tutorials and reflection that will each surface a particular CS concept. After sharing best practices, we will invite the audience to contribute their own experiences.

KEYWORDS

abstract strategy games, computational game theory, artificial intelligence, CS 1, CS 2, software engineering

1 SUMMARY

The goal of this panel is to introduce four educators who have employed abstract strategy games in their classrooms and research groups, and have them describe why and how they have used them to teach core computer science concepts. The intended audience is any high school and university instructor looking for ways to engage their students with authentic projects that touch on almost every aspect of the curriculum: abstraction, software engineering, databases, discrete mathematics, algorithms, parallel and distributed computing, human-computer interfaces, and artificial intelligence.

2 PANEL STRUCTURE

We will begin with five minutes to set the context of the panel. Then, each of the panelists will have ten minutes to share the ways in which they have used abstract strategy games in their instruction. These will be interactive, containing an introduction, micro-lesson and reflection. The remaining 30 minutes will be devoted to Q&A. Participants will be invited to unmute and share thoughts, concerns, questions, and any experiences they have had using abstract strategy games to teach computer science as well.

3 DAN GARCIA - GAMESCRAFTERS

Twenty years ago, I founded an undergraduate computational game theory research and development group at UC Berkeley called *GamesCrafters* [2]. The intellectual focus of the group is purely on small games that can be strongly solved [4]. New students are all asked to author, in any language, code to solve the Nim-like game *10-to-0-by-1-or-2* (on their turn, two players remove 1 or 2 coins from an initial pile of 10, and the first to get the pile to 0 wins). Through several subsequent assignments, they continue to refine their solver and add other games like generalized Tic-Tac-Toe.

At about five weeks in, the group shifts from being a class to a more traditional research group, and students work on their own projects that they propose. All are strongly encouraged to work in teams, and veterans serve as group leads (e.g., Back-end, Front-end, Theory, Solvers, etc). Typical projects are adding a new game with a text interface, adding a graphical interface to a game already created, optimizing a solver to use a new parallel technique, creating a more efficient database format, or doing analysis of a game using our tools. Every once in a while a student will invent a new game that can later be encoded, solved, and analyzed.

I do a fair bit of outreach to make sure my favorite (especially diverse) students know about the group and are encouraged to enroll. Overall, students learn how to work in a collaborative research environment, sharpen their large-code-base software skills, utilize their discrete mathematics, and explore their passion. My favorite testimonial was from a recent alumnus: *"I learned more about CS from my time in GamesCrafters than from all my classes!"*.

Dan Garcia is a Teaching Professor in the EECS department at UC Berkeley. He has worked with more than 500 students in his GamesCrafters group over twenty years.

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03.
https://doi.org/10.1145/3408877.3432572

4 IVONA BEZÁKOVÁ - PLAYER STRATEGIES FOR BOARD GAMES IN INTRO CS

In a joint project with colleagues James Heliotis and Sean Strout [1], we asked students in a CS2 course to design and implement their own player strategies for a specific board game; we have used abstract strategy games *Gobblet!* and *Quoridor* in the past.

The rationale behind the project is simple: (1) Board games are well-established and well-liked. We selected popular (but somewhat obscure) games to decrease the likelihood that students know them well; all students should have equal footing. (2) Highly-ranked games most likely do not have a clear winning strategy, so the strategy design is very open-ended. (3) Many board games nicely demonstrate several introductory computer science concepts (such as arrays, lists, stacks, and other data structures, as well as various types of search, including graph traversals). Therefore, they are a natural context for these topics. Students need to apply what they learned in class to this context, plus they have the option to go beyond if they want to improve their player's performance. (4) Watching our end-of-term *Battle Royale*, where the player strategies faced each other while the students (and their instructors) cheered them on, was definitely the highlight of the term! Beating the instructor's strategy was another cherished prize.

Ivona Bezáková, a lead on an NSF award titled "Multiplayer Board Game Strategies in the Introductory CS Curriculum," is a Professor in the Department of Computer Science at RIT.

5 ADAM BLANK - MATHEMATICAL THINKING, RECURSION, PARALLELISM

Games are a topic that students find very relatable that conveniently blends mathematical thinking and programming. They provide several opportunities throughout core courses in a CS curriculum to get students invested in difficult or unpalatable topics. I use games to get students invested in three core ideas: (1) programming is a *tool* for CS, (2) some problems are naturally recursive, (3) parallelism can lead to a significant speed-up on intractable problems.

The first module focuses on programming as a tool for CS. It starts with a recursive definition of what it means to win a combinatorial game such as Nim (N piles of coins, you may take M coins from any one pile on your turn). Then, using the definition, we solve 1-pile Nim analytically. After giving students a chance to attempt to solve N-pile Nim analytically, I redirect them to write a recursive program to solve the game instead.

The second module focuses on determining if a Tic-Tac-Toe board is winnable as an enumeration problem. Starting with a partially finished board reduces the search space enough for this to be feasible with a brute force algorithm. I've alternated between discussing the abstract algorithm and actually programming the solution during a lecture with students. Regardless of the format, this particular example has generated more excitement about enumeration problems than typical challenges (e.g., enumerating all strings of a particular length).

Finally, the third module focuses on abstract strategy games as a graph search problem. After teaching BFS, DFS, and Dijkstra's algorithm, we teach Minimax and Alpha-Beta. Then, students implement these algorithms in their own bot for a particular game. I've used both *Chess* and *Othello* in the past. The data structure course where this assignment has appeared also covers basic Fork-Join parallelism; as a result, we ask students to write a parallel version of Alpha-Beta called *Jamboree* [3] and run their code on a 36-core machine. When comparing performance, students realize that Jamboree is able to go to a further depth and generate better moves than just Alpha-Beta by itself! The assignment is also very open-ended; we provide them with poor board and move implementations and encourage them to do research on how to better represent these (e.g., bitboards).

Adam Blank is an Assistant Teaching Professor in the Computing and Mathematical Sciences Department at Caltech. They have used games in several courses at several institutions over the past eight years to motivate various concepts in Computer Science.

6 NEAL TERRELL - SOFTWARE ABSTRACTIONS AND GAME DESIGN

I use abstract strategy games to inspire programming and software engineering lessons in my courses. We begin with Tic-Tac-Toe, which is known to most students and easily learned by the rest. The game makes a great case study in CS1: during a live design-and-code exercise, I help students plan an abstract loop for playing the game in a text console; their conclusion is always some variant of *print* (display the game state), *input* (user chooses a move), *validate* (loop until a valid move is chosen), *apply* (update the game state with the chosen move), and *loop* (continue until the game is done). We talk about functional decomposition and translating each of the loop steps into a function. We also discuss game state representation (How many distinct states can a square be in? What other facts have to be remembered?), and separating the game state *data* from its *presentation* to the user.

We tackle harder games in CS2 and software engineering, and while the lessons vary with the language and grade level, the overall abstraction is the same: we program abstract games with a print, input, validate, and apply loop. As object-oriented principles are introduced, we design types to represent boards and moves, and use inheritance and polymorphism to build an abstract game-playing application. Design patterns like Strategy and Abstract Factory keep this abstract application extensible, and simplify implementations of more complex games like Chess. Finally, an instructor-provided implementation of a well-known game can be unit tested by students to improve their software verification skills, with awards for students who find deliberately-hidden errors in the implementation. Students love this; who doesn't enjoy correcting their teacher?

Neal Terrell is a Lecturer in the Computer Engineering and Computer Science Department at CSU Long Beach.

REFERENCES

[1] Ivona Bezáková, James E. Heliotis, and Sean Strout. 2013. Board game strategies in introductory computer science. In *The 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, Denver, CO, USA, March 6-9, 2013*, Tracy Camp, Paul T. Tymann, J. D. Dougherty, and Kris Nagel (Eds.). ACM, 17–22. https://doi.org/10.1145/2445196.2445210

[2] Dan Garcia. 2020. GamesCrafters. http://gamescrafters.berkeley.edu/. Accessed: 2020-08-23.

[3] Christopher Joerg and Bradley C. Kuszmaul. 1994. Massively Parallel Chess. In *In Proceedings of the Third DIMACS Parallel Implementation Challenge, Rutgers.*

[4] R. Bell and M. Cornelius. 1988. *Board Games Round the World: A Resource Book for Mathematical Investigations.* Cambridge University Press.

[5] Wikipedia. 2020. List of abstract strategy games. https://en.wikipedia.org/wiki/List_of_abstract_strategy_games. Accessed: 2020-08-23.